Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Randl

Hibridizacija genetskega algoritma za reševanje problema vozliščnega pokritja

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Ljubljana, 2017

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Randl

Hibridizacija genetskega algoritma za reševanje problema vozliščnega pokritja

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: dr. Andrej Brodnik

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V teoretičnem računalništvu lahko probleme razdelimo v različne razrede glede na njihovo zahtevnost. Posebej dobro znani so problemi v razredu NP (nedeterministično polinomski), med njimi so najzahtevnejši tako imenovani NP-polni problemi. Trenutno ne vemo, ali za te probleme obstaja deterministično polinomski algoritem. Kljub temu moramo omenjene probleme v praksi nekako rešiti in eden najpogostejših pristopov je z uporabo metahevristik.

V diplomski nalogi se osredotočite na problem vozliščnega pokritja ter implementirajte genetski metahevristični algoritem za njegovo reševanje. Nato hibridizirajte algoritem z iskanjem lokalnega optimuma. Slednje naredite preko vseh generacij, pri začetnih, pri osrednjih in pri končnih generacijah. Preučite učinke na kakovost rezultata in čas računanja glede na stopnjo hibridizacije in glede na število generacij.

IZJAVA O AVTORSTVU
DIPLOMSKEGA DELA

Spodaj podpisani Klemen Randl sem avtor diplomskega dela z naslovom: Hibridizacija genetskega algoritma za reševanje problema vozliščnega pokritja.

S svojim podpisom izjavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

Kazalo

Povzetek

Naslov: Hibridizacija genetskega algoritma za reševanje problema vozliščnega pokritja

V tej diplomski nalogi je obravnavan problem najmanjšega vozliščnega pokritja, ki je eden izmed zahtevnejših optimizacijskih problemov iz teorije grafov, spada namreč v kategorijo NP-težkih problemov. Verjetnost, da je mogoče optimalno rešitev zanj najti v polinomskem času, je zelo majhna, obstaja pa več načinov za iskanje približka optimalne rešitve. Tukaj je predstavljen način reševanja tega problema z genetskim algoritmom, ki se izkaže za precej učinkovitega, saj je z njegovo uporabo mogoče najti zelo dober približek optimalne rešitve v sprejemljivem času.

Z uvedbo hibridizacije v genetski algoritem se kakovost rešitve v določenih primerih močno izboljša, vendar se poveča tudi čas izvajanja algoritma, odvisno od tega, kako pogosto in na katerih delih algoritma se hibridizacija izvede. Izkaže se, da najboljšo rešitev algoritem najde, če je hibridizacija enakomerno razporejena, vendar za to porabi tudi največ časa. V primeru hibridizacije samo na sredini ali samo na koncu izvajanja algoritma je najdena rešitev za malenkost slabša, vendar algoritem do nje pride v veliko krajšem času. V primerih, ko je čas izvajanja algoritma pomemben, je takšna hibridizacija torej bolj smiselna od enakomerno porazdeljene.

S povečevanjem odstotka hibridizacije se kakovost rešitve sicer izboljšuje, ampak se hitro povečuje tudi porabljeni čas. Podobno se zgodi tudi v primeru povečevanja števila generacij (ustavitveni pogoj algoritma), vendar se izkaže, da je v primerih, ko je potrebna boljša rešitev, bolj smiselno povečati odstotek

hibridizacije kot pa število generacij, saj algoritem tako najde boljšo rešitev v krajšem času.

Ključne besede: vozliščno pokritje, genetski algoritem, hibridizacija, lokalna optimizacija.

Abstract

Title: Hybridisation of Genetic Algorithm for Vertex Cover Problem

In this bachelor's thesis, we discuss minimum vertex cover problem, which is one of the most complex problems in graph theory. It is a typical example of a NP-hard optimization problem. Finding an optimal solution in polynomial time is highly improbable, but there are many ways of finding an approximate solution to the problem. One of the options presented here is a use of a genetic algorithm, which turns out to be quite effective, since it is able to find a reasonably good solution in an acceptable amount of time.

In certain cases, the quality of solution is highly improved with the introduction of hybridization to the genetic algorithm, but the running time of the algorithm increases as well. The result depends on the section of the algorithm to which the hybridization is applied to, and how frequently it is used. As it turns out, the best solution is obtained with the periodically applied hybridization. However, the running time of the algorithm is the longest in this case. If hybridization is applied only in the middle or at the end of the run, the solution found is only slightly inferior, but the running time is much shorter. This type of hybridization is suitable for the cases in which the running time is a bigger concern than the quality of solution.

If the hybridization is used more frequently, the quality of solution increases, but so does the running time. The same happens when the number of generations (algorithm's termination condition) is increased. However, in the cases where the quality of solution is a priority, it is more sensible to increase the frequency of the hybridization than to increase the number of

generations because that way the algorithm finds better solutions in shorter time.

Keywords: vertex cover, genetic algorithm, hybridisation, local optimisation.

Poglavje 1

Uvod

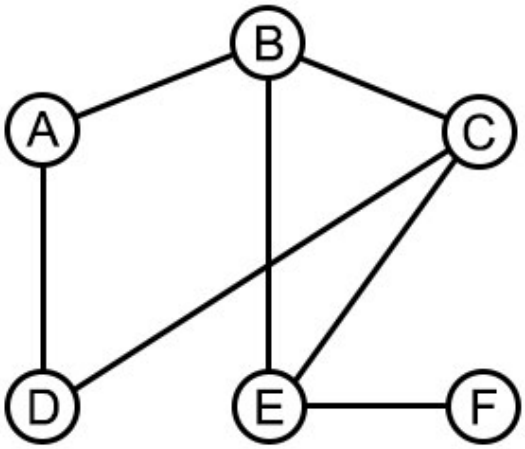
Teorija grafov je disciplina, ki se ukvarja z različnimi problemi, ki jih lahko opisujemo z grafi. Obstaja več kategorij takih problemov, nekatere izmed njih so na primer barvanje grafov, obhodi, pretoki in pokritja. Vozliščno pokritje je poseben primer pokritja grafa, pri katerem je treba z vozlišči pokriti vse povezave. S to problematiko se ukvarja problem najmanjšega vozliščnega pokritja, ki velja za enega izmed zahtevnejših problemov iz teorije grafov.

1.1 Vozliščno pokritje

Definicija 1.1 *Množica točk V' ($V' \subseteq V$) je vozliščno pokritje grafa $G = (V, E)$, če velja*

$$uv \in E \Rightarrow u \in V' \vee v \in V' \tag{1.1}$$

Vozliščno pokritje je takšna množica vozlišč grafa, ki pokrije vse povezave. To pomeni, da moramo, če hočemo najti vozliščno pokritje določenega grafa, najti takšno množico vozlišč, ki za vsako povezavo v grafu vsebuje najmanj eno krajišče. Poiskati vozliščno pokritje ni tako težko, saj je vozliščno pokritje lahko na primer kar množica vseh vozlišč grafa. Takšna rešitev je zelo neuporabna, saj nam ne pove veliko. Ponavadi je cilj poiskati čim manjše vozliščno pokritje grafa, torej čim manjše ali kar najmanjše možno število vozlišč, tako da še vedno velja definicija 1.1.



Slika 1.1: Preprost graf z najmanjšim vozliščnim pokritjem velikosti 3.

V primeru grafa na sliki 1.1, ki ima šest vozlišč in sedem povezav, lahko vidimo, da obstaja najmanjše vozliščno pokritje velikosti 3. Do tega lahko pridemo na dva načina. Prva možnost vključuje vozlišča A, C in E:

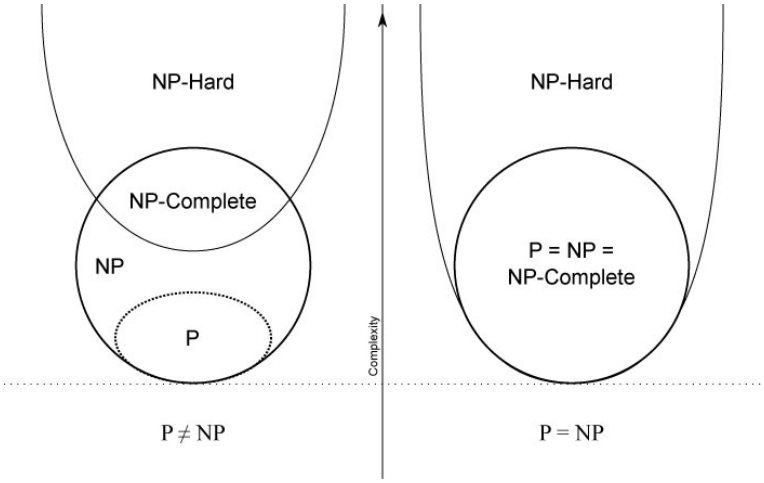
- vozlišče A pokrije povezavi AB in AD,
- vozlišče C pokrije povezave BC, CD in CE,
- vozlišče E pa pokrije povezavi BE in EF.

Ta tri vozlišča torej pokrijejo vseh sedem povezav. Obstaja tudi druga možnost, njena velikost je ravno tako 3. Ta možnost vsebuje vozlišča B, D in E:

- vozlišče B pokrije povezave AB, BC in BE,
- vozlišče D pokrije povezavi AD in CD,
- vozlišče E pa pokrije povezavi CE in EF.

Poleg dveh naštetih vozliščnih pokritij obstaja za isti graf na sliki 1.1 še nekaj drugih vozliščnih pokritij, ki pa niso najmanjša možna, na primer: vozlišča A, B, C in E, ki tvorijo vozliščno pokritje velikosti 4 in ravno tako pokrivajo vse povezave, ali pa vozlišča A, B, C, D in E, ki tvorijo vozliščno pokritje velikosti 5. Vozliščno pokritje tvori tudi na primer kar vseh šest vozlišč grafa, ampak, kot rečeno, je ponavadi cilj najti čim manjše vozliščno pokritje.

Velikosti najmanjšega možnega vozliščnega pokritja grafa pravimo številka vozliščnega pokritja grafa (*ang. vertex cover number* [1]) in jo označimo z malo grško črko tau - $\tau(G)$. Če je najmanjše vozliščno pokritje grafa velikosti n, to pomeni, da za ta graf ne obstaja nobeno vozliščno pokritje velikosti manjše od n. Najti čim manjše vozliščno pokritje, še posebej za večje grafe, se izkaže za zelo zahtevno nalogo in zato spada problem vozliščnega pokritja med NP-polne probleme.



Slika 1.2: Eulerjev diagram množic P-, NP-, NP-težkih in NP-polnih problemov (vir slike: [2]).

1.2 NP-problemi

V teoriji računske zahtevnosti poznamo več razredov, s katerimi opisujemo težavnost oziroma zahtevnost posameznih problemov. Če je problem rešljiv v polinomskem času z deterministčnim Turingovim strojem, potem spada v zahtevnostni razred P. V razred problemov NP (*nondeterministic, polynomial time*) spadajo tisti odločitveni problemi, ki jih v polinomskem času lahko reši nedeterministični Turingov stroj in katerih rešitev lahko preverimo deterministično v polinomskem času.

Množica problemov zahtevnostnega razreda P je podmnožica množice NP-problemov ($P \subseteq NP$). Eno večjih odprtih vprašanj v računalništvu je, ali sta množici P-problemov in NP-problemov pravzaprav enaki ($P = NP$). Če se izkaže, da sta, bi to pomenilo, da je vsak problem, katerega rešitev lahko preverimo v polinomskem času, tudi rešljiv v polinomskem času. Na sliki 1.2 lahko vidimo razliko med odnosi med zahtevnostnimi razredi. Leva stran velja, če predpostavljamo, da množici P-problemov in NP-problemov nista enaki, desna stran pa velja, če predpostavljamo, da sta.

V razred NP-težkih (*ang. NP-hard*) problemov spadajo tisti problemi, na

katere se lahko v polinomskem času pretvori vse NP-probleme. Drugače povedano, NP-težki problemi so tisti problemi, ki so vsaj tako težki kot najtežji NP-problemi, ni pa nujno, da sami spadajo v razred NP-problemov. V razred NP-polnih (*ang. NP-complete*) problemov pa spadajo tisti problemi, ki so hkrati NP ter NP-težki.

Problem vozliščnega pokritja spada v razred NP-polnih problemov, kot je leta 1972 pokazal Richard Karp [3]. Stephen Cook je leta 1971 pokazal, da je problem izpolnjivosti (*ang. boolean satisfiability problem*) NP-poln [4]. Karp pa je njegov dokaz uporabil, da je pokazal, da je problem izpolnjivosti v polinomskem času mogoče reducirati na 21 drugih problemov iz kombinatorike in teorije grafov, med katerimi je tudi problem vozliščnega pokritja.

1.3 Odločitveni in optimizacijski problem

Z odločitvenim problemom vozliščnega pokritja se soočimo, če imamo graf G in število k ter nas zanima, ali za graf G obstaja vozliščno pokritje velikosti k ali manj. Ta odločitveni problem je NP-poln [3], zato je malo verjetno, da bomo do rešitve prišli v polinomskem času (to bi pomenilo, da smo dokazali $P = NP$ - glej poglavje 1.2). Tak problem lahko rešimo s požrešno metodo v eksponentnem času, če pa je vrednost k dovolj majhna, lahko do rešitve pridemo tudi v polinomskem času.

Pri optimizacijskih problemih pa izmed vseh možnih rešitev iščemo ravno najboljšo. Problem najmanjšega vozliščnega pokritja je klasični optimizacijski problem, saj nas izmed vseh vozliščnih pokritij grafa zanima le najmanjše. Ker pa je problem najmanjšega vozliščnega pokritja NP-težak, do rešitve prav tako ne moremo priti v polinomskem času. Lahko pa pridemo do približka rešitve oziroma optimalno rešitev aproksimiramo.

Deterministično lahko problem najmanjšega vozliščnega pokritja aproksimiramo največ do faktorja $1,3606$, kot sta leta 2005 dokazala I. Dinur in S. Safra [5]. Pokazala sta, da je aproksimacija minimalnega vozliščnega pokritja na faktor $1,3606$ ali manj NP-težak problem pod predpostavko $P \neq NP$.

1.4 Aproksimacijski algoritmi

Aproksimacijski algoritmi za konstantni faktor (*ang. constant-factor approximation algorithms*) so taki algoritmi, ki imajo razmerje aproksimacije omejeno s konstanto. Za problem najmanjšega vozliščnega pokritja obstaja preprost algoritem (odkrila sta ga Fanica Gavril in Mihalis Yannakakis [6]), ki optimalno rešitev aproksimira na faktor 2 (algoritem 1). To pomeni, da je pridobljeno vozliščno pokritje v najslabšem primeru dvakrat večje od optimalnega.

Algoritem 1 Aproksimacija-tockovnega-pokritja(G)

1: $C = \emptyset$
2: $E' = G.E$
3: **while** $E' \neq \emptyset$ **do**
4: Naj bo (u, v) povezava iz E'
5: $C \leftarrow C \cup (u, v)$
6: Iz E' odstrani vse povezave, ki se dotikajo vozlišča u
7: Iz E' odstrani vse povezave, ki se dotikajo vozlišča v
8: **end while**
9: **return** C

Najboljši trenutno znani algoritem za problem najmanjšega vozliščnega pokritja ima faktor aproksimacije $2 - \Theta(\frac{1}{\sqrt{log|V|}})$. Ta algoritem je leta 2005 predstavil George Karakostas [7].

Če vzamemo v obzir še gostoto grafa, lahko najmanjše vozliščno pokritje aproksimiramo na faktor $\frac{2}{1+\delta}$ za graf z gostoto δ. Gostota je podana kot $\delta = \frac{|E|}{|V|(|V|-1)}$, kjer je E število povezav grafa in V število vozlišč grafa. Tako za prazni graf velja $\delta = 0$ in za polni graf velja $\delta = 1$, kar pomeni, da je faktor aproksimacije za polni graf še vedno 2, vendar se ta faktor zmanjšuje proti 1 za grafe z manjšo gostoto. To sta s svojim aproksimacijskim algoritmom pokazala Marek Karpinski in Alexander Zelikovsky [8].

1.5 Struktura naloge

V prvem poglavju tega diplomskega dela je predstavljena narava problema vozliščnega pokritja. V drugem poglavju je predstavljen drugačen pristop k problemu - reševanje problema z uporabo genetskega algoritma. Genetski algoritem je eden izmed evolucijskih algoritmov, ki pri svojem delovanju uporabljajo elemente biološke evolucije, kot so na primer izbira, križanje in mutacija. Opisani so sestavni elementi takšnega algoritma in njegovo delovanje. V tretjem poglavju so predstavljeni rezultati testiranja genetskega algoritma, hibridiziranega z implementacijo lokalne optimizacije, ki se izkaže za zelo učinkovitega pri aproksimaciji najmanjšega vozliščnega pokritja. V četrtem poglavju sledijo povzetek rezultatov in zaključki.

Poglavje 2

Evolucijski pristop

Evolucijski algoritem je metahevristični optimizacijski algoritem, ki pri svojem delovanju uporablja koncepte biološke evolucije. Je poseben pristop k reševanju optimizacijskih problemov, ki prinese zadovoljive rezultate z manj računskega truda kot na primer deterministične metode.

Genetski algoritem je najbolj razširjen tip evolucijskega algoritma, ki deluje po principu naravnega izbora. Do rešitve pride tako, da skozi generacije razvija oziroma izboljšuje prvotno rešitev. Prvotna rešitev oziroma prva generacija je populacija kandidatov (osebkov), ki so navadno generirani naključno. Vsak kandidat ima genski zapis in indeks. Genski zapis predstavlja potencialno rešitev in je navadno zakodiran v binarni niz, možni pa so tudi drugi načini kodiranja. Indeks je kazalnik kakovosti genskega zapisa. Določi ga funkcija ustreznosti (*ang. fitness function*) in pokaže, kako dober je kandidat. Boljši kandidati imajo večji indeks in so boljši približek optimalne rešitve problema.

Nad kandidati iz prve generacije izvedemo biološke operacije, kot so na primer izbira, križanje in mutacija, s čimer pridobimo drugo generacijo, za katero računamo, da je rahlo boljša od prve. Postopek ponovimo, da dobimo tretjo generacijo, za katero je zaželeno, da je še boljša od druge, in tako naprej, vse dokler ne dobimo dovolj dobre rešitve oziroma dokler pogoj za ustavitev algoritma ni dosežen. Okvirno delovanje algoritma je prikazano na

sliki 2.1.

2.1 Genetski algoritem za vozliščno pokritje

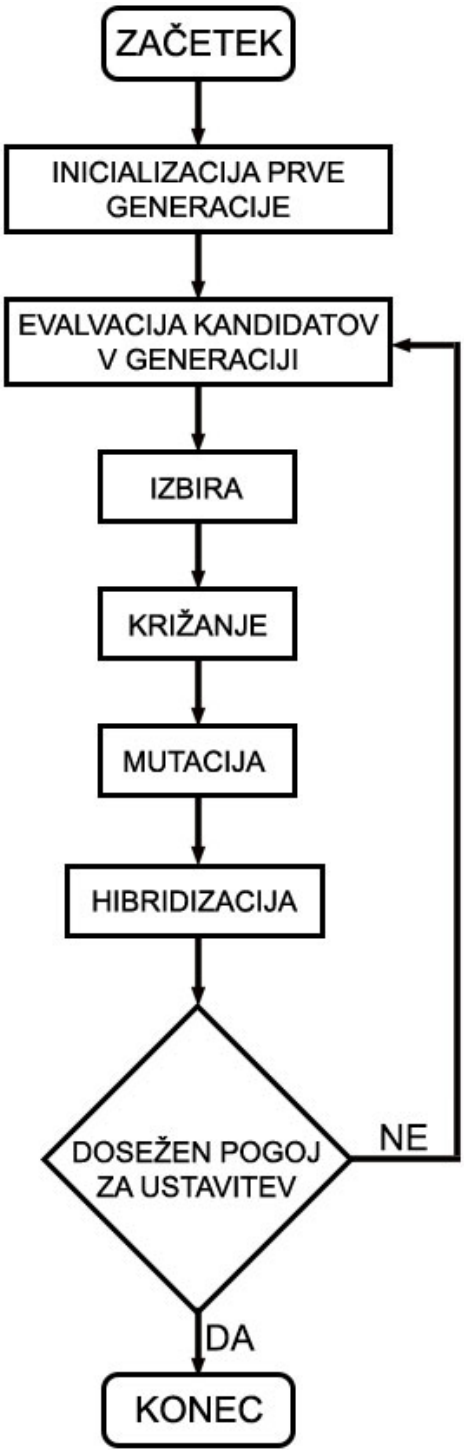
Genetski algoritem je primeren za iskanje približka optimalne rešitve za problem najmanjšega vozliščnega pokritja. Potencialne kandidate za rešitev lahko enostavno predstavimo z binarnim nizom, zaradi česar jih lahko elegantno križamo in mutiramo. Kakovost genskega zapisa lahko učinkovito preverimo s funkcijo ustreznosti. Genetski algoritem lahko tudi hibridiziramo, tako da mu dodamo deterministično komponento, ki občasno izvede lokalno iskanje nad določenimi kandidati v populaciji in tako hitreje najde lokalni minimum kandidata ter na splošno izboljša končno rešitev.

2.2 Elementi genetskega algoritma

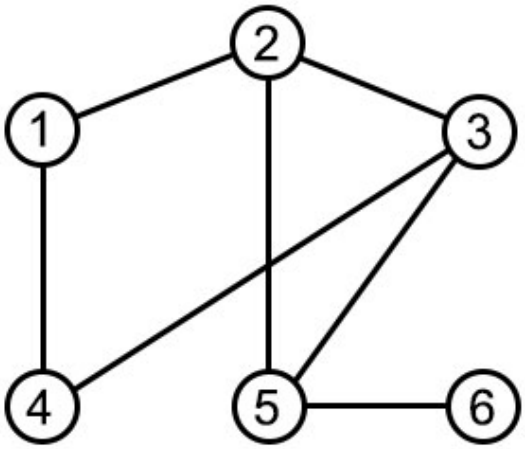
Genetski algoritem ima več ključnih sestavnih delov in parametrov. Najti je treba način za predstavitev kandidatov, torej njihove lastnosti opisati z genskim zapisom (informacija o tem, katera vozlišča grafa kandidat vsebuje). V prvi generaciji so lastnosti kandidatov določene naključno. Nad kandidati v generaciji se nato zaporedoma izvaja več genetskih operacij, kot prikazuje slika 2.1. Ta postopek se ponavlja iterativno skozi večje število generacij, dokler pogoj za ustavitev algoritma ni izpolnjen.

2.2.1 Genski zapis

Pri genetskem algoritmu moramo najti način, kako kandidata za rešitev problema enostavno predstavimo oziroma njegove lastnosti zakodiramo v genski zapis tako, da lahko na njem izvajamo genetske operacije. Informacijo o tem, katera vozlišča grafa kandidat vsebuje, lahko predstavimo z binarnim nizom. Ta niz vsebuje toliko bitov, kolikor je vozlišč v grafu. Če so vozlišča predstavljena s številkami od 1 do n, kjer je n število vseh vozlišč, potem i-ti bit v



Slika 2.1: Diagram poteka genetskega algoritma.



Slika 2.2: Primer grafa, ki ima vozlišča označena s številkami.

genskem zapisu nosi informacijo o tem ali kandidat i-to vozlišče vsebuje ali ne.

Če bi hoteli na primeru grafa s slike 2.2 predstaviti kandidata, ki vsebuje vozlišča $\{1, 2, 3, 5\}$, bi imel ta kandidat genski zapis 111010:

- ker kandidat vsebuje vozlišče 1, je prvi bit 1,
- ker kandidat vsebuje vozlišče 2, je drugi bit 1,
- ker kandidat vsebuje vozlišče 3, je tretji bit 1,
- ker kandidat vozlišča 4 ne vsebuje, je četrti bit 0,
- ker kandidat vsebuje vozlišče 5, je peti bit 1,
- ker kandidat vozlišča 6 ne vsebuje, je šesti bit 0.

2.2.2 Velikost generacije

Za genetski algoritem moramo določiti, koliko kandidatov bo vsebovala vsaka generacija. Če je generacija premajhna, pride do premalo raznolikosti in algoritem prehitro konvergira k lokalnemu minimumu ter ne išče več v širino. Če je velikost generacije prevelika, pa lahko algoritem postane časovno preveč potraten in postane skoraj naključno iskanje. Velikost generacije je načeloma odvisna od tipa problema, ki ga rešujemo, ampak navadno šteje od nekaj sto do nekaj tisoč kandidatov.

V genetski algoritem lahko uvedemo koncept elite. Elita je določen majhen delež populacije (na primer najboljših 5 %), ki neposredno napreduje v naslednjo generacijo. Ideja je, da se zelo dobri kandidati ohranijo v naslednji generaciji, saj se lahko zgodi, da bi se z uporabo genetskih operacij, kot je na primer mutacija, pokvarili oziroma bi izgubili kakovost (bi se oddaljili od optimalne rešitve) in bi bila zaradi tega naslednja generacija slabša od prejšnje. Pri izbiri velikosti elite moramo biti pazljivi, saj lahko v primeru prevelike elite algoritem zelo hitro stagnira (izboljšava v kakovosti generacij je zanemarljiva ali pa je sploh ni).

Prvo generacijo kandidatov navadno generiramo naključno, kar pomeni, da vsem kandidatom v prvi generaciji naključno določimo genski zapis. Vsak kandidat v prvi generaciji torej predstavlja naključno množico točk grafa in je zato verjetnost, da bo kandidat dober, zelo majhna. V prvi generaciji ne moremo pričakovati dobrih rešitev. Komaj skozi veliko število generacij se ta prvotna populacija s pomočjo genetskih operatorjev izboljšuje in tako približa optimalni rešitvi.

2.2.3 Funkcija ustreznosti

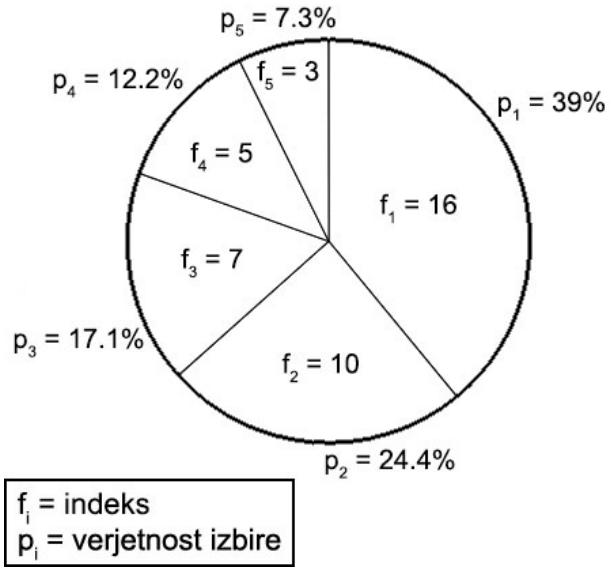
Pri genetskem algoritmu je funkcija ustreznosti ena izmed pomembnejših komponent. Funkcija ustreznosti posameznega kandidata oceni na podlagi njegovega genskega zapisa in nam pove, kako dober je kandidat. Zelo pomembno je, da ta funkcija poda dobro oceno kakovosti kandidata in da je

časovno nezahtevna, saj se v algoritmu izvaja zelo pogosto (oceniti mora vsakega kandidata v vsaki generaciji). Na podlagi te ocene (indeksa) algoritem izbira kandidate, ki tvorijo naslednjo generacijo. Če funkcija ustreznosti ne poda realne ocene, se generacije s časom ne izboljšujejo in posledično genetski algoritem najde slab rezultat.

Najbolj enostaven način evalvacije kandidata je, da mu s funkcijo ustreznosti priredimo številčni indeks. Večji kot je indeks, boljši je kandidat. Optimalna rešitev mora imeti največji indeks izmed vseh možnih kandidatov. Preprost kazalnik kakovosti kandidata je že to, ali točke, predstavljene z genskim zapisom kandidata, tvorijo vozliščno pokritje ali ne. Če točke, predstavljene z genskim zapisom, grafa vozliščno ne pokrijejo, je kandidat manj primeren. Če kandidat predstavlja vozliščno pokritje, je vprašanje, kako dobro je to vozliščno pokritje. Manj vozlišč kot vsebuje vozliščno pokritje, boljše je in je zato smiselno, da kandidat, ki predstavlja vozliščno pokritje manjše velikosti, dobi večji indeks ter ima posledično večjo možnost, da bo uporabljen pri generiranju nove generacije. To pa še ne pomeni, da slabših kandidatov ne upoštevamo. Tudi kandidati s slabšim indeksom so pomembni, in sicer za raznolikost. Če bi uporabljali samo kandidate z najboljšimi indeksi, potem bi algoritem zelo hitro konvergiral k lokalnemu minimumu. Tudi kandidati s slabšim indeksom morajo torej imeti nekaj možnosti za izbiro pri generiranju nove generacije.

2.2.4 Izbira

Izbira je operacija, ki izmed kandidatov v trenutni generaciji izbere tiste, ki bodo uporabljeni pri generiranju naslednje generacije. V primeru genetskega algoritma za iskanje vozliščnega pokritja, kjer je ustreznost posameznega kandidata navadno predstavljena z indeksom (številko), je zelo preprost in pogosto uporabljan način izbire premo sorazmerni izbor (*ang. fitness proportionate selection*). Pri takem načinu izbire je možnost, da bo posamezni kandidat izbran, sorazmerna z njegovim indeksom, in sicer $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, kjer je N število vseh kandidatov ter f indeks (ustreznost) posameznega kandi-



Slika 2.3: Premo sorazmerni izbor.

data. V primeru na sliki 2.3 je vsota vseh indeksov 41, zato je verjetnost, da bo izbran na primer prvi kandidat z indeksom 16, enaka $p_1 = \frac{16}{41} = 0,39$.

Še en preprost način izbire je izbor s turnirjem (*ang. tournament selection*). Pri taki izbiri določimo velikost turnirja, torej koliko kandidatov bo naključno izbranih iz trenutne generacije za borbo na turnirju. Za napredovanje v naslednjo generacijo je izbran tisti kandidat, ki ima največji indeks izmed vseh kandidatov na turnirju. Če je velikost turnirja 1, to pomeni, da izbira postane naključna (vedno je izbran naključni kandidat, ne glede na indeks), če pa je velikost turnirja enaka kar velikosti generacije, pa bo vedno izbran kandidat z največjim indeksom v generaciji. Ključno je najti ravno pravo velikost turnirja, tako da so indeksi pri bolj primernih kandidatih ustrezno upoštevani, hkrati pa dobijo tudi manj primerni kandidati z manjšimi indeksi možnost za izbiro.

Obstajajo tudi drugi tipi izbire, ampak ker se izbira v genetskem algoritmu izvede zelo pogosto, ni priporočljivo uporabiti zahtevnih načinov izbire, saj se lahko čas, ki ga genetski algoritem porabi za iskanje rešitve, zelo hitro podaljša, če izbira ni opravljena v smiselnem času.

Slika 2.4: Različni načini križanja.

2.2.5 Križanje

Križanje deluje tako, da iz trenutne generacije z operacijo izbire dobimo dva kandidata (starša) in ju križamo med seboj, da dobimo dva potomca, ki ju vključimo v naslednjo generacijo. Pred vključitvijo gresta potomca še skozi proces mutacije - glej poglavje 2.2.6. Pri križanju uporabimo del genskega zapisa prvega starša in del genskega zapisa drugega starša, da tvorimo genski zapis potomcev.

Obstaja več načinov križanja, ki se med seboj večinoma razlikujejo po točki križanja. Trije različni načini križanja so predstavljeni na sliki 2.4. V prvem primeru na sliki je točka križanja ena sama, torej je pri prvem potomcu prvi del genskega zapisa (do točke križanja) podedovan od prvega starša, drugi del (od točke križanja naprej) pa od drugega starša. Pri drugem potomcu je prvi del od drugega starša, drugi del pa od prvega. Tak način križanja se imenuje križanje preko ene točke (*ang. single-point crossover*). To točko lahko postavimo na sredino genskega zapisa in tako potomca vsebujeta enak delež genskega zapisa obeh staršev, lahko pa točko naključno določimo in tako deleža genskih zapisov staršev ne bosta enaka.

V drugem primeru na sliki 2.4 se križanje opravi preko dveh točk (*ang. two-point crossover*). Pri takem načinu križanja starša križamo na dveh različnih točkah v genskem zapisu. Prvi potomec od prvega starša podeduje prvi del genskega zapisa (do prve točke križanja) ter tretji del genskega zapisa (od druge točke križanja naprej), od drugega starša pa podeduje drugi del (med prvo in drugo točko križanja). Drugi potomec pa ravno obratno.

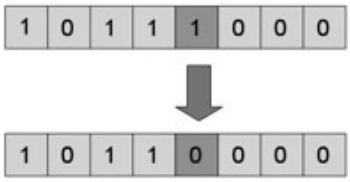
Enakomerno križanje (*ang. uniform crossover*) ima več točk križanja, ampak je prispevani delež genskega zapisa obeh staršev približno enak. Tak način križanja je predstavljen v tretjem primeru na sliki 2.4. Točke križanja so določene naključno, ampak velja, da ima vsak bit v genskem zapisu potomcev 50% verjetnost, da bo podedovan od prvega starša, in 50% verjetnost, da bo podedovan od drugega.

Pri polovičnem enakomernem križanju (*ang. half uniform crossover*) se križa natanko polovica tistih bitov, ki se med staršema razlikujejo. Tisti biti, ki so enaki pri obeh starših, ostanejo enaki tudi pri obeh potomcih, križajo se samo tisti biti, ki se med staršema razlikujejo. Najprej se izračuna Hammingova razdalja med staršema (število bitov, ki se med njima razlikeujejo). Nato prvi potomec polovico teh bitov podeduje od prvega starša, polovico pa od drugega, drugi potomec pa obratno.

2.2.6 Mutacija

Mutacija je operacija, s katero spremenimo naključni bit (ali več bitov) v genskem zapisu kandidata. Je pomemben del vsakega genetskega algoritma, saj ohranja gensko raznolikost kandidatov. Verjetnost, da posamezen bit v genskem zapisu mutira, ne sme biti prevelika, saj lahko tako iskanje postane preveč naključno. Mutacija že zelo majhnega števila bitov v genskem zapisu lahko popolnoma spremeni kakovost kandidata in tako lahko genetski algoritem bolj učinkovito razišče iskalni prostor in posledično najde boljše rešitve.

Eden izmed enostavnejših načinov mutacije je negacija bita. To za primer vozliščnega pokritja pomeni, da če kandidat predstavlja množico vozlišč



Slika 2.5: Mutacija z negacijo enega bita.

grafa, ki vsebuje določeno vozlišče, predstavljeno z bitom, ki bo mutiran, tega vozlišča po mutaciji ne vsebuje več (in obratno). V primeru s slike 2.5 kandidat pred mutacijo predstavlja množico točk, ki vozlišče, predstavljeno s petim bitom, vsebuje, saj je ta bit postavljen na 1. Po mutaciji kandidat tega vozlišča ne vsebuje več.

Za probleme, kjer genskega zapisa kandidata ne moremo predstaviti z binarnim nizom, je treba uporabiti drugačen tip mutacije, ampak za problem vozliščnega pokritja je najbolj smiseln ravno zgoraj opisani način z negacijo bita.

2.2.7 Ustavitveni pogoj

Genetskemu algoritmu je treba določiti pogoj za ustavitev. Od pogoja je odvisno, koliko časa se bo algoritem izvajal in kako dobro rešitev bo našel. Navadno najbolj osnoven pogoj je fiksno maksimalno število generacij algoritma. Ko algoritem doseže vnaprej določeno število generacij, se prekine in vrne rešitev, do katere je tekom tega števila generacij prišel. Če algoritem omejimo s takim pogojem, je treba to maksimalno število generacij izbrati tako, da ne bo premajhno, saj tako algoritem ne bo imel časa približati se optimalni rešitvi. Če pa je število preveliko, lahko od neke točke naprej nove generacije algoritma ne predstavljajo smiselne izboljšave nad prejšnjo generacijo.

To je lahko tudi eden izmed pogojev za ustavitev algoritma. Vsaka nova generacija je navadno boljša od prejšnje, ampak te razlike so največje

na začetku algoritma in se tekom števila generacij vedno bolj manjšajo. Določimo lahko najmanjši faktor izboljšave kakovosti rešitve in ko algoritem pride do točke, ko je izboljšava vsake nove generacije nad prejšnjo manjša od tega faktorja, se algoritem ustavi. Nesmiselno je proizvajati veliko število novih generacij, če niso nič boljše od prejšnje.

Še en možen pogoj za ustavitev algoritma je lahko kriterij minimalne kakovosti rešitve. Ko algoritem doseže rešitev, ki je za dano situacijo dovolj dobra, se ustavi. Tak kriterij pride v poštev na primer, ko ne iščemo čim boljšega vozliščnega pokritja, ampak le dovolj dobro vozliščno pokritje. Če je na primer v grafu 500 vozlišč in algoritmu damo kriterij 450 vozlišč, se bo algoritem ustavil takoj, ko bo eden izmed kandidatov tvoril vozliščno pokritje velikosti 450 ali manj. Ta kandidat bo za dano situacijo dovolj dober, četudi je optimalno vozliščno pokritje veliko na primer 400 vozlišč in bi se algoritem tej rešitvi lahko veliko bolj približal.

Pogoj za ustavitev algoritma je lahko tudi kaj drugega. Eden izmed pogojev je lahko na primer čas. Ko poteče določen časovni interval, se algoritem ustavi in vrne rešitev, do katere je v danem času prišel. V nekaterih primerih lahko algoritem ustavimo tudi ročno, ko smo zadovoljni z doseženo kakovostjo rešitve.

Vse naštete pogoje za ustavitev genetskega algoritma lahko tudi kombiniramo in tako zagotovimo, da bo algoritem našel dovolj ustrezno rešitev znotraj podanih mej.

2.3 Hibridizacija

Eden izmed načinov za izboljševanje genetskega algoritma je hibridizacija, kar pomeni, da v proces genetskega algoritma uvedemo občasno lokalno optimizacijo. Za določen majhen odstotek vseh generacij se izvede sistematično lokalno iskanje nad kandidati, kar privede do hitrejšega odkritja lokalnega minimuma. Hibridizacijo navadno določimo v odstotkih, na primer 10% hibridizacija na algoritmu, ki teče 1.000 generacij, bi pomenilo, da se lokalna

optimizacija izvede nad 100 generacijami. Nad točno katerimi generacijami se hibridizacija izvede, je drug parameter. Hibridizacijo lahko uporabimo na različnih odsekih med izvajanjem genetskega algoritma, in sicer se lahko lokalna optimizacija izvede nad:

- začetnimi generacijami algoritma,
- sredinskimi generacijami algoritma,
- končnimi generacijami algoritma,
- enakomerno porazdeljenimi generacijami čez celoten potek algoritma,
- naključno določenimi generacijami.

Na omenjenem primeru s 1.000 generacijami in z 10% hibridizacijo bi hibridizacija nad začetnimi generacijami pomenila, da se lokalno optimizira zgolj prvih 100 generacij. Pri hibridizaciji na sredini bi se lokalna optimizacija izvedla nad generacijami od 451 do 550, pri končni hibridizaciji pa nad generacijami od 901 do 1.000. Pri enakomerno razporejeni hibridizaciji bi se lokalno optimizirala vsaka deseta generacija, pri naključni pa bi se 100 generacij za optimizacijo določilo naključno.

2.3.1 Vpliv hibridizacije

Hibridizacija pozitivno vpliva na rezultat genetskega algoritma. V članku [9] je pokazano, da že uvedba majhnega odstotka hibridizacije (10%) močno izboljša rešitev. Ko se odstotek hibridizacije povečuje, se kakovost rešitve navadno še dodatno izboljšuje, ampak se ta izboljšava postopoma manjša, povečuje pa se čas izvajanja algoritma. V članku so objavljeni rezultati testiranj za naključno hibridizacijo, hibridizacijo na začetku in hibridizacijo na koncu. Najboljšo rešitev prinese hibridizacija na koncu.

Podobna testiranja so opisana v [10], kjer so bili opravljeni testi za enakomerno porazdeljeno hibridizacijo ter hibridizacijo na začetku, sredini in na koncu. Rezultati pokažejo, da najboljše rezultate prinese enakomerno

porazdeljena hibridizacija, ki v [9] ni bila preizkušena. Velik napredek v kakovosti rešitve je prav tako opazen pri uvedbi majhnega odstotka hibridizacije (5%). Rezultati so pokazali, da se kakovost rešitve tu bistveno ne izboljša s povečevanjem odstotka hibridizacije. Presenetljivo se kakovost rešitve pri 30- in večodstotni hibridizaciji celo slabša.

Poglavje 3

Algoritem in rezultati

V tem poglavju je predstavljen naš genetski algoritem za iskanje približka najmanjšega vozliščnega pokritja. Predstavljeni so parametri algoritma in okvirno delovanje ter tudi rezultati, ki jih algoritem prinese, in njihova analiza. Pri testiranju smo se najbolj osredotočili na vpliv različnih načinov hibridizacije na rezultat genetskega algoritma in na njegov pretečeni čas.

3.1 Okolje za testiranje

Okolje za testiranje je skupek strojne in programske opreme, ki je bila uporabljena za pridobitev rezultatov, predstavljenih v tem poglavju.

Vsi testi so bili izvedeni na računalniku s procesorjem Intel® Core™ i5-4460 s štirimi jedri in s štirimi nitmi, ki teče na frekvenci med 3.20 ter 3.40 GHz. Uporabljen je bil operacijski sistem Windows 10, različice 1607 (kodno ime *Redstone*). Algoritem je bil implementiran v programskem jeziku Java z razvojnim okoljem *NetBeans 8.2 Patch 1*, uporabljen je bil *Java™ SE Development Kit 8, Update 111*.

ime grafa	št. vozlišč	najmanjše vozliščno pokritje	št. primerov
frb30-15-mis	450	420	5
frb35-17-mis	595	560	5
frb40-19-mis	760	720	5
frb45-21-mis	945	900	5
frb50-23-mis	1150	1100	5
frb53-24-mis	1272	1219	5
frb56-25-mis	1400	1344	5
frb59-26-mis	1534	1475	5

Tabela 3.1: Primeri grafov, na katerih je bil algoritem testiran.

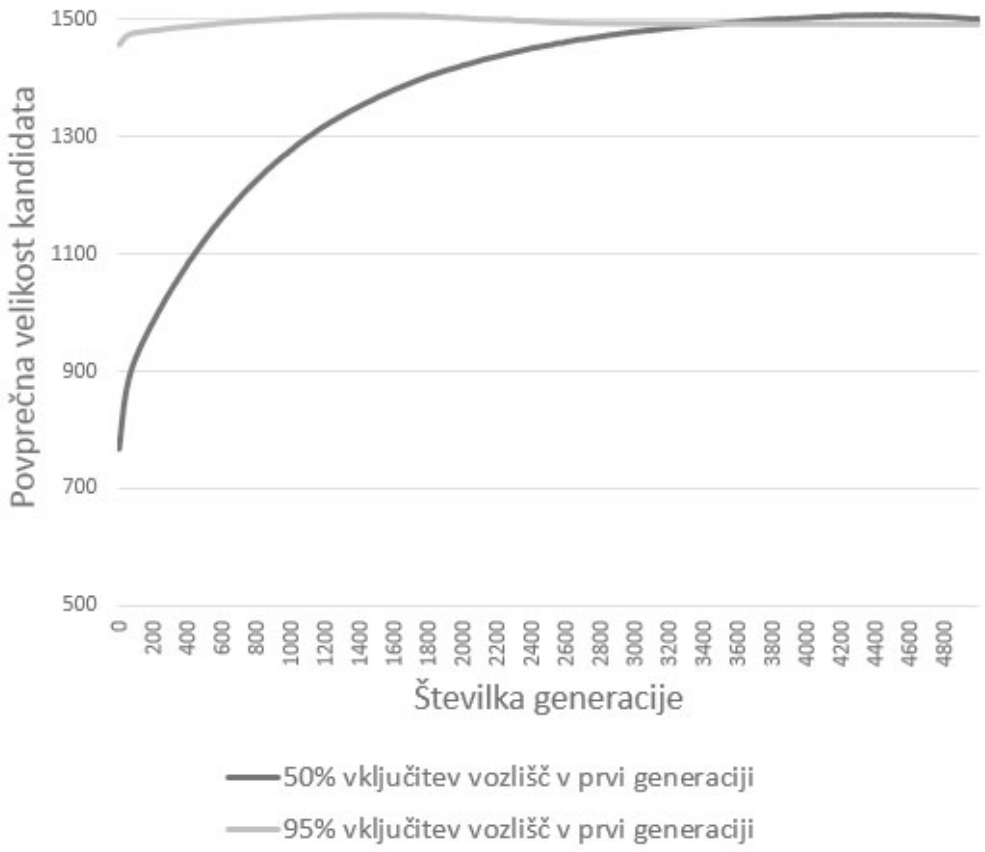
3.2 Primeri grafov

Primeri grafov, ki so bili uporabljeni za testiranje algoritma, so bili preneseni s spletne strani [11], kjer je objavljenih 40 primerov grafov, po 5 primerov za vsako velikost (tabela 3.1).

Za optimizacijo parametrov algoritma in za razvijanje samega algoritma ter testiranje njegovega delovanja so bili uporabljeni vsi grafi, rezultati, predstavljeni v nadaljevanju tega poglavja, pa so pridobljeni s poganjanjem algoritma večinoma na največjih grafih, velikosti 1534 vozlišč (*frb59-26-mis*).

3.3 Implementacija genetskega algoritma

Velikost generacije pri našem genetskem algoritmu je nastavljena na 100, velikost elite pa na 5. Pri naključnem generiranju genskih zapisov kandidatov prve generacije je verjetnost, da je posamezni bit nastavljen na 1, enaka 95%, možnost, da je posamezni bit 0, pa 5%. Vsako posamezno vozlišče ima torej večjo možnost, da bo vključeno med vozlišča naključno generiranega kandidata prve generacije, kot pa da ne bo, kar pomeni, da imajo že v prvi generaciji kandidati večje število vozlišč. Zaradi tega algoritem prej najde kandidate, ki predstavljajo vozliščno pokritje grafa in se zaradi tega algoritem



Slika 3.1: Povprečna velikost kandidatov skozi generacije za primer *frb59-26-5.mis*.

začne prej približevati optimalni rešitvi, kot če bi bila verjetnost na primer 50%.

To je vidno na grafu na sliki 3.1, kjer smo beležili povprečno velikost kandidatov skozi generacije algoritma za oba primera. Graf za verjetnost 95% se začne prej spuščati (okoli generacije 1.500) kot graf za verjetnost 50%, ki se začne spuščati komaj okoli generacije 4.500. To se zgodi, ker za vse primere, na katerih smo naš algoritem testirali, velja, da imajo optimalno rešitev precej veliko (velikost najmanjšega vozliščnega pokritja je v primerjavi z velikostjo grafa še vedno dokaj velika), zato algoritem v primeru 50%

naključnega generiranja prve generacije potrebuje veliko število generacij, da sploh najde kandidate, ki predstavljajo vozliščno pokritje, in ima zato manj časa, da jih izboljšuje, ter je posledično rešitev, ki jo algoritem najde, slabša.

Za redkejše grafe, kjer je velikost optimalnega vozliščnega pokritja približno polovica velikosti celotnega grafa, bi bilo smiselno prvo generacijo naključno generirati s 50% verjetnostjo vključitve posameznega vozlišča.

3.3.1 Funkcija ustreznosti

Funkcija ustreznosti deluje tako, da kandidatom, ki predstavljajo vozliščno pokritje, priredi boljši indeks kot kandidatom, katerih množica vozlišč za podani graf ne predstavlja vozliščnega pokritja. Pri kandidatih, ki predstavljajo vozliščno pokritje, je indeks večji, če vsebujejo manj vozlišč, pri kandidatih, ki ne predstavljajo vozliščnega pokritja, pa je indeks večji, če vsebujejo več vozlišč (slika 3.2). Takšen način ocenjevanja kakovosti kandidatov ni ravno najbolj optimalen, saj v določenih primerih kandidata ne oceni čisto realno. Ta pomanjkljivost se najbolj pokaže pri kandidatih, ki ne predstavljajo vozliščnega pokritja, ampak so zelo blizu optimalni rešitvi. Ti kandidati imajo vedno manjši indeks od kandidatov, ki predstavljajo vozliščno pokritje grafa, četudi z neuporabno velikim številom vozlišč. Do tega pride, ker je kakovost kandidatov, ki vozliščnega pokritja ne predstavljajo, zelo težko dobro oceniti. Ker mora biti funkcija ustreznosti implementirana na čim bolj preprost način (glej poglavje 2.2.3), smo se zadovoljili s to poenostavljeno izvedbo.

Čeprav torej funkcija ustreznosti ni najbolj optimalna, ampak je preprosta in ni časovno potratna, algoritem deluje v pravi smeri ter prinese dobre rezultate.

Delovanje funkcije ustreznosti je predstavljeno z algoritmom 2. Funkcija kandidatu priredi indeks na podlagi tega, ali kandidat tvori vozliščno pokritje ali ne. Kot je razvidno tudi s slike 3.2, imajo kandidati, ki predstavljajo vozliščno pokritje, večji indeks od kandidatov, ki ne tvorijo vozliščnega pokritja. Koliko večja je ta razlika, je odvisno od faktorja f. Za naš algoritem je bil ta faktor nastavljen na vrednost 10.

Slika 3.2: Grafična predstavitev ocenjevanja kandidatov v našem algoritmu.

Algoritem 2 Funkcija ustreznosti

1: $A \leftarrow$ Kandidat
2: $n \leftarrow$ Število vozlišč v grafu
3: $f \leftarrow 10$
4: **if** vozliscnoPokritje(A) **then**
5: $\quad A.indeks \leftarrow f * n - A.velikost$
6: **else**
7: $\quad A.indeks \leftarrow A.velikost$
8: **end if**

3.3.2 Izbira

V našem algoritmu je implementiran premo sorazmeren izbor. Večji kot je indeks kandidata, večja verjetnost je, da bo kandidat uporabljen pri generiranju naslednje generacije. Postopek izbire določi dva različna kandidata iz trenutne generacije (starša), ki gresta čez proces križanja ter mutacije in nastala potomca gresta nato v naslednjo generacijo. To algoritem počne toliko časa, dokler naslednja generacija ni polna.

Z algoritmom 3 je predstavljeno delovanje premo sorazmernega izbora. Kot argument vzame seznam indeksov vseh kandidatov v generaciji, urejen po velikosti od največjega do najmanjšega. Kot rezultat vrne zaporedno številko kandidata, ki je bil izbran. Ta kandidat se nato uporabi kot eden izmed

Algoritem 3 Premo sorazmerni izbor

1: $sez\_index \leftarrow$ Seznam indeksov po velikosti od max do min
2: $sum \leftarrow$ Seštevek vseh indeksov
3: $rand \leftarrow$ Naključno celo število ($1 \leq rand \leq sum$)
4: $i \leftarrow 0$
5: **while** $rand > 0$ **do**
6: $\quad rand \leftarrow rand - sez\_index[i]$
7: $\quad i \leftarrow i + 1$
8: **end while**
9: **return** i

staršev pri križanju. Podrobnejši opis takega načina izbora je v poglavju 2.2.4.

3.3.3 Križanje

Križanje se v algoritmu ne izvede čisto vsakič. Verjetnost, da bosta izbrana kandidata križana, je 90%. Če do križanja pride, se naključno določi ena točka na genskem zapisu kandidatov in na tej točki se ta dva starša križata. Nastala potomca gresta naprej na proces mutacije. V 10% primerov, ko se križanje ne izvede, gresta izbrana kandidata neposredno na proces mutacije.

Algoritem 4 Križanje

1: $A \leftarrow$ Binarni niz genskega zapisa prvega starša
2: $B \leftarrow$ Binarni niz genskega zapisa drugega starša
3: $meja \leftarrow$ Naključno celo število ($1 \leq meja < A.length$)
4: **for** $i = meja \ldots (A.length - 1)$ **do**
5: $\quad temp \leftarrow A[i]$
6: $\quad A[i] \leftarrow B[i]$
7: $\quad B[i] \leftarrow temp$
8: **end for**
9: **return** A, B

Postopek križanja je prikazan v algoritmu 4. Kot argument sta podana genska zapisa obeh staršev. V algoritmu se starša križata preko ene točke, rezultat pa sta dva nova genska zapisa. To sta potomca, ki napredujeta v naslednji korak - mutacijo.

3.3.4 Mutacija

Mutacija se ravno tako ne izvede čisto vsakič. Izvede se z verjetnostjo 90%. V 10% primerov, ko se mutacija ne izvede, gre kandidat kar v naslednjo generacijo. Ko pa do mutacije pride, se naključno izbere en bit iz genskega zapisa kandidata, ki se negira (če je izbran bit 0, po mutaciji postane 1 in obratno) in nato gre kandidat v naslednjo generacijo. Postopek mutacije je opisan z algoritmom 5. Ko kandidat pride v novo generacijo, ga funkcija ustreznosti ponovno ovrednoti ter mu priredi nov indeks.

Algoritem 5 Mutacija

1: $A \leftarrow$ Binarni niz genskega zapisa kandidata
2: $i \leftarrow$ Naključno celo število $(0 \leq i < A.length)$
3: NegirajBit($A[i]$)
4: **return** A

3.3.5 Ustavitveni pogoj

Kot ustavitveni pogoj algoritma je podano le največje število generacij, torej koliko generacij se genetski algoritem izvaja. V prvem testu, ko smo testirali vpliv hibridizacije, je bil ustavitveni pogoj nastavljen na 5.000 generacij. V drugem testu pa smo testirali tudi vpliv števila generacij na dobljeno rešitev ter na čas izvajanja algoritma. Pri tem testiranju smo poleg pogoja 5.000 generacij testirali še pogoj 10.000 in pogoj 15.000 generacij.

3.3.6 Hibridizacija

V algoritem smo implementirali tudi hibridizacijo. Ugotavljali smo, kako hibridizacija na različnih odsekih (na začetku, sredini, koncu in enakomerno porazdeljena) vpliva na kakovost pridobljene rešitve ter koliko se zaradi tega razlikuje čas izvajanja algoritma. Spreminjali smo tudi stopnjo hibridizacije. Nekateri viri kažejo na to, da v določenih primerih že majhna stopnja hibridizacije prinese veliko izboljšavo rešitve [9] in da nekateri testi prikazujejo najboljše rezultate z ozirom na čas izvajanja algoritma za stopnjo hibridizacije okoli 5% [10]. To smo preverjali tudi na našem genetskem algoritmu, tako da smo algoritem testirali za tri različne stopnje hibridizacije, in sicer 5%, 10% in 15%.

Hibridizacijo med algoritmom občasno izvajamo nad vsemi kandidati v populaciji, in sicer z lokalno optimizacijo, katere osnova je delček algoritma, prestavljenega v [12]. Lokalna optimizacija našega genetskega algoritma deluje tako, da kandidatu, ki predstavlja vozliščno pokritje grafa, postopoma sistematično odstranjujemo vozlišča tako, da še vedno ostane vozliščno pokritje, ampak s čim manjšim številom vozlišč. Ko ne moremo odstraniti nobenega vozlišča več, preverimo, ali ima katero izmed vozlišč v tej novi množici točk natanko enega soseda izven pokritja. Če tako vozlišče obstaja, algoritem poskuša ti dve vozlišči zamenjati in nastali množici vozlišč ponovno sistematično odstranjevati odvečna vozlišča tako, da na koncu pride do lokalnega minimuma.

Velja omeniti, da implementirana lokalna optimizacija deluje samo nad kandidati, ki predstavljajo vozliščno pokritje. Za kandidata, ki predstavlja množico vozlišč, ki ne tvorijo vozliščnega pokritja, algoritem ne more učinkovito poiskati dobrega lokalnega minimuma in zato lokalna optimizacija nad takimi kandidati ne deluje.

Kako deluje lokalna optimizacija, je prikazano z algoritmom 6. Procedura 1 se rekurzivno izvaja nad kandidati, ki imajo odstranljiva vozlišča. Vozlišče je odstranljivo, če ga lahko iz vozliščnega pokritja odstranimo, tako da kandidat še vedno predstavlja vozliščno pokritje. Procedura 2 poskuša zamenjati

Algoritem 6 Lokalna optimizacija

```
1: A ← Kandidat
2: Procedura1(A)
3: Procedura2(A)
4: return A
5: procedure PROCEDURA1
6:     A ← Kandidat
7:     n ← Število vozlišč grafa
8:     if A.nimaOdstranjivihVozlisc then
9:         return A
10:    end if
11:    for i = 0 . . . (n − 1) do
12:        if JeMogoceOdstraniti(A[i]) then
13:            st[i] ← Število vozlišč, ki jih je mogoče odstraniti, če odstranimo vozlišče i
14:        end if
15:    end for
16:    Max(st) ← Za to vozlišče obstaja največ odstranljivih vozlišč
17:    A.odstrani(Max(st))
18:    return Procedura1(A)
19: end procedure
20: procedure PROCEDURA2
21:    A ← Kandidat
22:    v ← Vozlišče, ki ima natanko enega soseda izven vozliščnega pokritja
23:    u ← Edini sosed vozlišča v izven vozliščnega pokritja
24:    A.odstrani(v)
25:    A.dodaj(u)
26:    return Procedura1(A)
27: end procedure
```

vozlišče v vozliščnem pokritju z edinim sosedom izven pokritja ter nato nad rezultatom izvede še proceduro 1.

3.4 Rezultati testiranja

Vsi rezultati, predstavljeni v tem poglavju, so bili pridobljeni tako, da se je za posamezni graf algoritem izvedel neodvisno 100-krat zaporedoma z istimi parametri. Pri tem smo merili:

- povprečno velikost vozliščnega pokritja, ki ga je vrnil algoritem, in
- povprečen čas trajanja algoritma.

3.4.1 Vpliv hibridizacije

V prvem testu smo preverjali vpliv različnih parametrov hibridizacije na genetski algoritem. Pogoj za ustavitev algoritma je bil za vse primere nastavljen na 5.000 generacij, spreminjali smo le odstotek hibridizacije (brez hibridizacije, 5% hibridizacija, 10% hibridizacija in 15% hibridizacija) ter odsek algoritma, kjer se hibridizacija izvede (na začetku, na sredini, na koncu in enakomerno čez celoten potek algoritma). To nam je pokazalo, kakšen vpliv imajo ti parametri na kakovost rešitve in kako se spremeni čas, ki ga algoritem potrebuje, da do te rešitve pride.

Algoritem smo testirali na vseh 40 podanih primerih grafov in rezultati so bili za vse uporabljene grafe zelo podobni. V tem poglavju so predstavljani samo rezultati za največje grafe (5 različnih grafov, vsak s 1534 vozlišči in optimalnim vozliščnim pokritjem velikosti 1475 - *frb59-26-mis*). Za izhodišče smo vzeli rezultate, ki jih genetski algoritem najde brez hibridizacije. Ti rezultati so predstavljeni v tabeli 3.2.

Rezultati nam povedo, da naš genetski algoritem najde precej dobro rešitev že brez hibridizacije. V povprečju za vseh pet grafov algoritem od optimalne rešitve odstopa le za malo več kot en odstotek. To pomeni, da v povprečju algoritem vrne rešitev, ki vsebuje 16 oziroma 17 vozlišč več kot

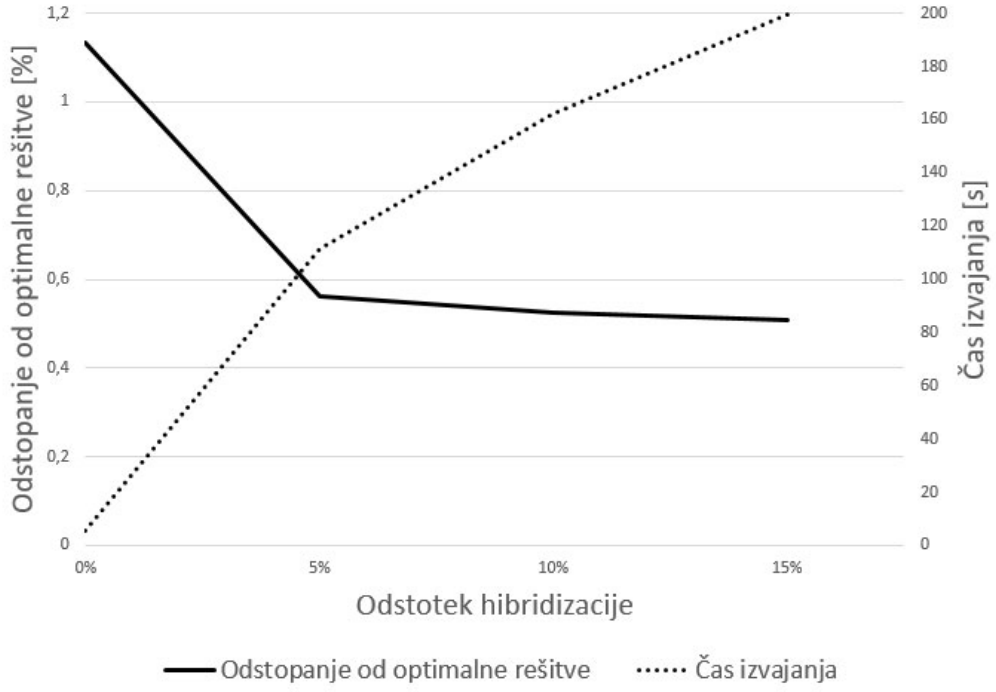
	velikost rešive		čas [s]		
graf	povprečje	odklon	povprečje	odklon	odstopanje [%]
frb59-26-1.mis	1491,49	1,65	5,42	0,23	1,1180
frb59-26-2.mis	1491,50	1,62	5,71	0,27	1,1186
frb59-26-3.mis	1491,98	1,49	5,72	0,23	1,1512
frb59-26-4.mis	1491,62	1,48	5,74	0,21	1,1268
frb59-26-5.mis	1491,82	1,60	5,71	0,26	1,1403

Tabela 3.2: Rezultati genetskega algoritma brez hibridizacije za 5 grafov velikosti 1534 z optimalnim vozliščnim pokritjem velikosti 1475.

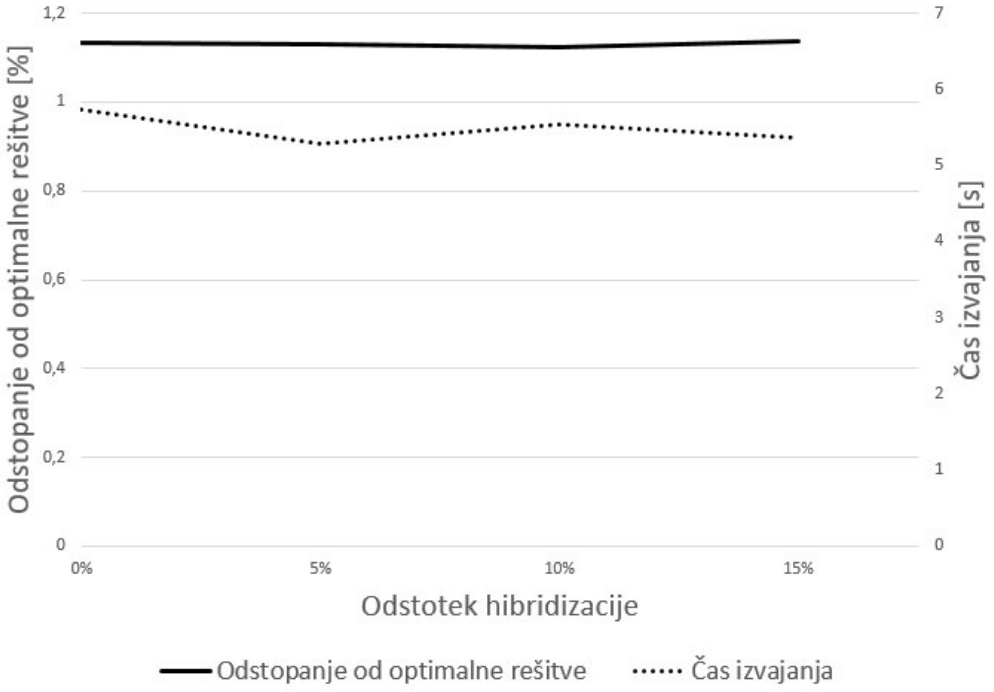
minimalno vozliščno pokritje grafa, ki je velikosti 1475. Algoritem za tako rešitev v povprečju porabi malo manj kot 6 sekund časa.

V nadaljevanju bodo predstavljeni rezultati, ki jih naš genetski algoritem najde, ko vključimo hibridizacijo. V teh testih smo spreminjali dva parametra, in sicer odstotek hibridizacije ter odsek algoritma, na katerem je bila hibridizacija uporabljena. Za vsak par teh parametrov smo algoritem neodvisno pognali 100-krat za vsakega izmed največjih petih grafov. V nadaljevanju je z grafi na slikah 3.3, 3.4, 3.5 in 3.6 prikazana trendna črta za kakovost rešitev ter za čas, ki ga je algoritem potreboval za različne vrednosti omenjenih parametrov.

Na grafu na sliki 3.3 vidimo, kako se naš algoritem obnaša, ko uvedemo enakomerno razporejeno hibridizacijo. Že za majhen odstotek optimiziranih generacij (5%) se kakovost rezultatov močno poveča. Optimalni rešitvi se rezultati približajo že na 0,56 odstotka, ampak se s tem močno poveča tudi porabljeni čas, ki strmo naraste na približno 110 sekund. Ko odstotek hibridizacije spremenimo na 10% se kakovost rešitve izboljša minimalno v primerjavi s 5% hibridizacijo, porabljeni čas pa se vseeno zelo poveča. Podobno se zgodi s 15% hibridizacijo. Rezultati se optimalni rešitvi sicer vedno bolj približajo (na okoli 0,51 odstotka), ampak algoritem za to porabi toliko časa, da se to v večini primerov ne izplača.



Slika 3.3: Enakomerno porazdeljena hibridizacija.



Slika 3.4: Izvajanje hibridizacije samo na začetku.

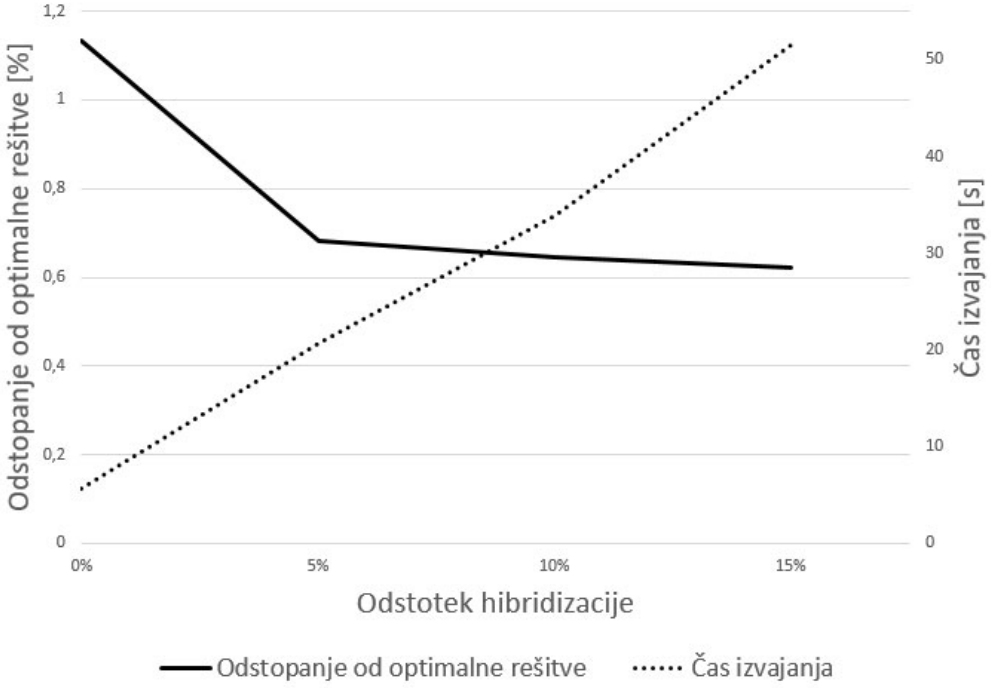
V primerjavi z drugimi tipi hibridizacije algoritem za enakomerno razporejeno hibridizacijo porabi veliko več časa kot na primer za hibridizacijo samo na koncu ali samo na sredini. Razlog za to je najverjetneje v tem, da lokalna optimizacija vozliščnih pokritij z veliko vozlišči traja veliko časa. Algoritem začne komaj po določenem številu generacij odkrivati tiste kandidate, ki tvorijo vozliščna pokritja in prve generacije takih vozliščnih pokritij so zelo daleč od lokalnega minimuma, zato ima del algoritma, ki izvaja lokalno optimizacijo, s takimi kandidati veliko dela ter zanje posledično porabi več časa. Ker se lokalna optimizacija izvaja enakomerno čez vse generacije, se algoritem takim prvim generacijam vozliščnih pokritij ne more izogniti. Rešitev, ki jih tak tip hibridizacije prinese, je sicer rahlo boljša od ostalih, ampak je razlika tako zanemarljivo majhna, da je cena, ki jo za to plačamo (veliko večji čas), v večini primerov neupravičena.

Ko smo hibridizacijo izvajali samo na začetnih generacijah, smo prišli do

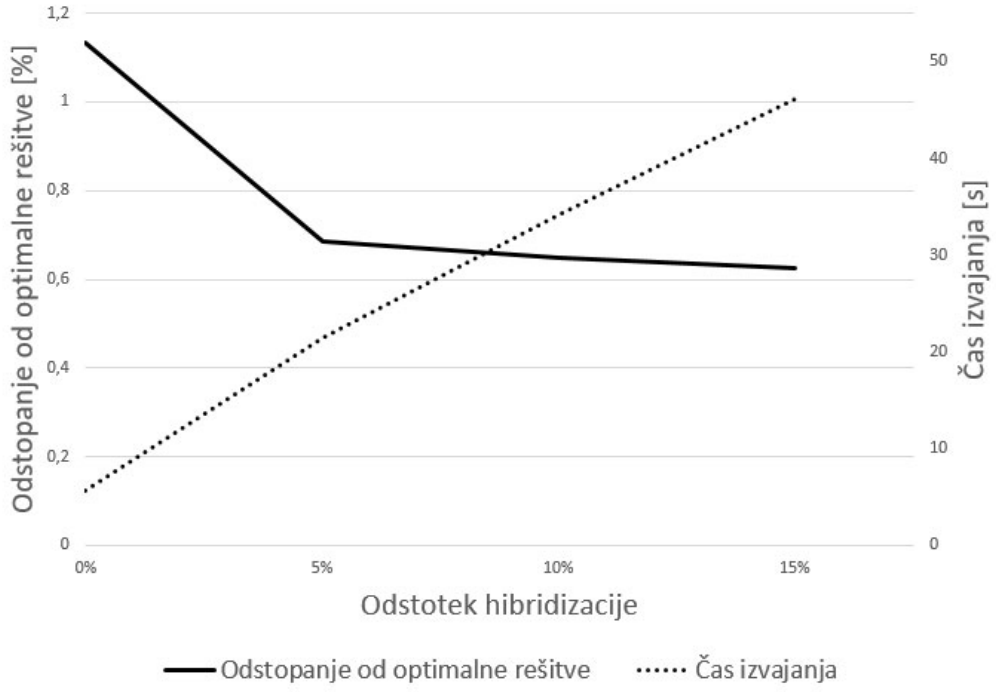
rezultatov, ki nam pokažejo, da tak tip hibridizacije nima učinka. Kot je razvidno z grafa na sliki 3.4 se kakovost rešitve in čas pri različnih odstotkih hibridizacije spremeni zgolj minimalno. Razlog je v tem, da lokalna optimizacija ne deluje za kandidate, ki ne predstavljajo vozliščnega pokritja. In ker se prvi kandidati, ki vozliščno pokritje predstavljajo, pojavijo komaj v poznejših generacijah, ko se lokalna optimizacija ne izvaja več, je posledica to, da se lokalna optimizacija tudi v zgodnjih generacijah v bistvu ne izvaja in smo zato dobili skoraj identične rezultate kot pri testu brez hibridizacije. Izkaže se, da je s takšno lokalno optimizacijo, kot je implementirana v našem algoritmu, hibridizacija začetnih generacij nesmiselna.

Če smo hibridizacijo izvajali na sredini (graf na sliki 3.5), smo dobili zelo podobne rezultate kot pri testih, ko smo hibridizacijo izvajali samo na koncu (graf na sliki 3.6). V obeh primerih je vidna velika izboljšava, ko se uvede 5% hibridizacija. Rezultati se v primeru hibridizacije na sredini optimalni rešitvi približajo na 0,68 odstotka, pri hibridizaciji na koncu pa na 0,69 odstotka. Za razliko od enakomerno porazdeljene hibridizacije pa tu ni opaziti tako velike razlike v času. Čas se je v obeh primerih povečal na približno 21 sekund, kar je še sprejemljivo. Ko odstotek hibridizacije povečamo na 10%, se pričakovano izboljša kakovost rešitve (razlika do optimalne rešitve je v obeh primerih okoli 0,65 odstotka) ter poveča čas izvajanja algoritma na približno 34 sekund v obeh primerih. Če odstotek hibridizacije povečamo še na 15%, se kakovost rešitve ponovno izboljša in je zdaj 0,62 odstotka od optimalne rešitve v obeh primerih, s tem da v primeru hibridizacije na sredini čas bolj naraste (na 51 sekund) kot pri hibridizaciji na koncu (46 sekund).

Opazimo torej, da hibridizacija na sredini v primeru 5% hibridizacije prinese rahlo boljše rezultate v malenkost krajšem času kot hibridizacija na koncu. Pri 10% hibridizaciji razlike skoraj ni, pri 15% hibridizaciji pa enako dobro rešitev v malo boljšem času najde hibridizacija na koncu. Tako dobre rešitve, kot jo prinese enakomerno razporejena hibridizacija, še vedno ne najde nobeden izmed teh dveh tipov hibridizacij, sta pa ta tipa hibridizacije zato kar precej bolj časovno ekonomična.



Slika 3.5: Izvajanje hibridizacije samo na sredini.



Slika 3.6: Izvajanje hibridizacije samo na koncu.

N	odstotek hibridizacije							
	0%		5%		10%		15%	
	D [%]	t [s]	D [%]	t [s]	D [%]	t [s]	D [%]	t [s]
5.000	1,16	5,75	0,69	22,39	0,65	35,37	0,62	46,16
10.000	1,05	12,41	0,66	44,63	0,63	61,98	0,61	82,07
15.000	0,98	19,78	0,64	58,17	0,63	97,97	0,59	126,57

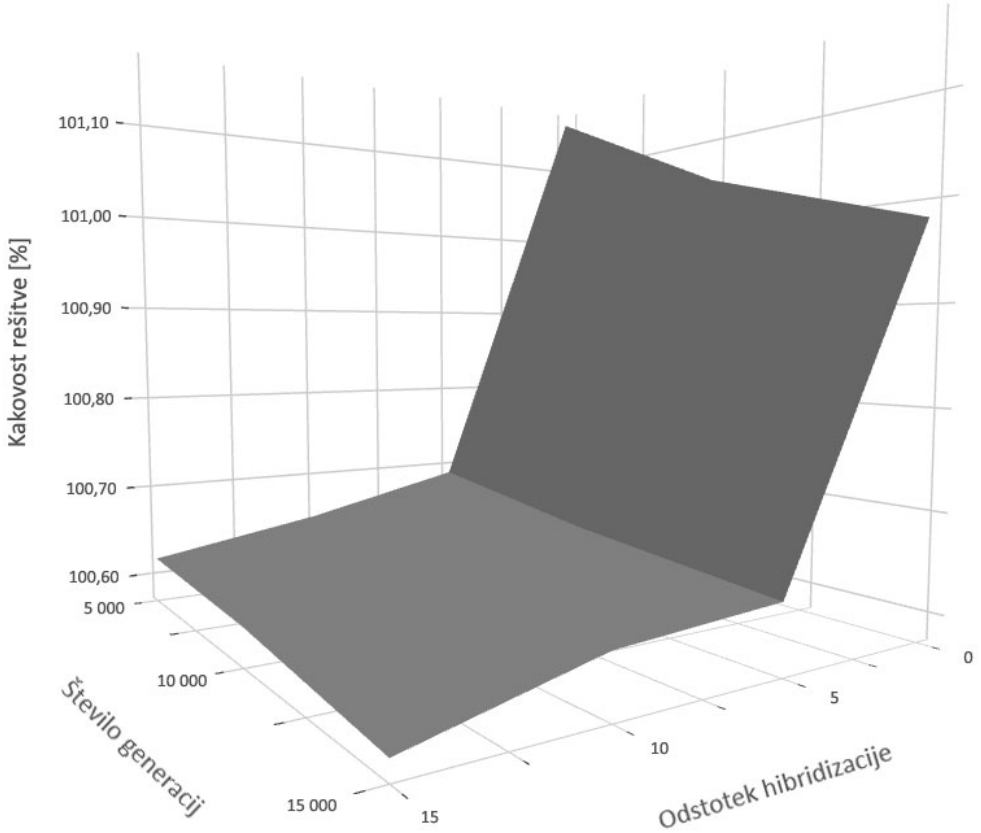
N - ustavitveni pogoj (število generacij), D - odstopanje od optimalne rešitve, t - čas

Tabela 3.3: Vpliv odstotka hibridizacije in števila generacij na kakovost rešitve ter potrebni čas.

3.4.2 Vpliv ustavitvenega pogoja

Pri drugem testu smo ugotavljali vpliv ustavitvenega pogoja na algoritem pri različnih odstotkih hibridizacije. Ta test smo izvedli samo na enem grafu (*frb59-26-5.mis* - 1534 vozlišč, najmanjše vozliščno pokritje velikosti 1475). Skozi celoten test je bila uporabljena hibridizacija samo na končnih generacijah, medtem ko smo spreminjali odstotek hibridizacije (0%, 5%, 10% in 15%) in maksimalno število generacij kot ustavitveni pogoj algoritma (5.000, 10.000 in 15.000 generacij). Rezultati tega testa so predstavljeni v tabeli 3.3.

Kot smo ugotovili že v prejšnjem poskusu, se kakovost rešitve izboljšuje, če povečujemo odstotek hibridizacije. Po pričakovanjih je opazna tudi izboljšava rešitve, če algoritmu dopustimo večje število generacij. To je vidno na sliki 3.7. Ugotovili pa smo, da ima večanje odstotka hibridizacije večji vpliv kot večanje števila generacij. Algoritem je v primeru 5% hibridizacije v 5.000 generacijah našel rešitev, ki se od optimalne razlikuje za 0,69 odstotka in je za to porabil 22,39 sekunde. Če smo povečali število generacij na 15.000, se je rešitev izboljšala za 0,05 odstotka, porabljeni čas pa se je povečal za 35,78 sekunde. Če pa namesto števila generacij povečamo hibridizacijo, dobimo rešitev, ki je za 0,07 odstotka boljša, čas pa se poveča le za 23,77 sekunde. Zaključimo lahko, da se bolj splača povišati odstotek generacij, nad



Slika 3.7: Odvisnost kakovosti rešitve od odstotka hibridizacije ter števila generacij.

katerimi se izvede optimizacija, kot pa povečati število generacij.

Poglavje 4

Zaključek

Cilj te diplomske naloge je bil predstaviti evolucijski pristop k reševanju optimizacijskega problema najmanjšega vozliščnega pokritja z uporabo genetskega algoritma. Algoritem smo implementirali in ga izboljšali z uvedbo hibridizacije. Raziskovali smo vpliv parametrov hibridizacije in vpliv ustavitvenega pogoja (števila generacij) na potek algoritma ter prišli do zanimivih rezultatov, ki so bili do neke mere pričakovani.

Naš algoritem je že brez uvedbe hibridizacije prinašal kar dobre rezultate v spodobnem času. Z uporabo lokalne optimizacije smo nato algoritem hibridizirali, kar je v splošnem privedlo do boljših rezultatov in povzročilo daljši čas izvajanja algoritma.

Kot nekateri drugi viri [9,10] smo tudi mi ugotovili, da že majhen odstotek hibridizacije znatno izboljša kakovost rešitve. Če ta odstotek povečujemo, se kakovost rešitve še dodatno izboljšuje, vendar to pomeni, da algoritem potrebuje vedno več časa, da do te rešitve pride. Pri povečevanju odstotka hibridizacije se čas povečuje približno linearno, kakovost rešitve pa se izboljšuje vedno manj.

Hibridizacijo smo v prvem testu preizkušali na različnih odsekih algoritma. Tako kot Tuma v [10] smo tudi mi ugotovili, da najboljšo rešitev prinese enakomerno razporejena hibridizacija. Rahlo presenetljiva je bila velika količina časa, ki ga je v takem primeru algoritem za izračun rešitve pora-

bil. Tudi hibridizacija na sredini ali na koncu izvajanja algoritma se izkaže za precej dobro izbiro. Kakovost rešitve je sicer rahlo slabša kot pri enakomerni porazdelitvi hibridizacije, ampak je zato algoritem veliko bolj časovno ekonomičen. Prišli smo do zaključka, da je pri določanju parametrov hibridizacije treba sklepati kompromise. Izbire, ki bi bila univerzalno optimalna, ni. Če potrebujemo rešitev, ki se čim bolj približa globalnemu optimumu in smo zanjo pripravljeni žrtvovati veliko časa, bomo izbrali enakomerno razporejeno hibridizacijo. Če pa je čas ključnega pomena, se bomo odločili za hibridizacijo na sredini ali na koncu, saj je tudi taka rešitev relativno zelo dobra. Podobno je z odstotkom hibridizacije. Lokalno optimizacijo izvajamo pogosteje, če potrebujemo boljšo rešitev ne glede na čas, in redkeje, če se lahko zadovoljimo z rešitvijo slabše kakovosti, pridobljeno v krajšem času. Več podrobnosti o tem je mogoče najti v poglavju 3.4.1.

V drugem testu smo kot spremenljivko uvedli še ustavitveni pogoj (število generacij). Prišli smo do ugotovitve, da se kakovost rešitve sicer izboljšuje, če povečujemo število generacij, ampak je bolje povečevati odstotek hibridizacije, saj tako dobimo boljše rešitve v krajšem času. Več podrobnosti je v poglavju 3.4.2.

Dele algoritma, ki smo ga za potrebe tega diplomskega dela razvili, je mogoče še dodatno izboljšati za potrebe pridobitve bolj specifičnih rezultatov. Algoritem bi gotovo deloval bolje, če bi funkcija ustreznosti natančneje ocenjevala kandidate, ki ne predstavljajo vozliščnega pokritja. Take kandidate je zelo težko objektivno oceniti, še posebej zato, ker mora biti funkcija ustreznosti časovno nezahtevna. Verjetno bi se izplačalo razviti boljšo implementacijo, saj je ta funkcija ena izmed ključnih komponent genetskega algoritma. Ena izmed možnih izboljšav je primerjava kandidatov, ki ne predstavljajo vozliščnega pokritja grafa s kandidati, ki vozliščno pokritje predstavljajo in imajo velik indeks. Če je razlika med dvema takima kandidatoma dovolj majhna, bi to pomenilo, da je kakovost kandidata, ki sicer ne predstavlja vozliščnega pokritja, dobra. Zato bi mu ta izboljšana funkcija ustreznosti priredila večji indeks kot naša funkcija ustreznosti.

Opravilo bi se lahko še obširnejše testiranje vseh parametrov genetskega algoritma, saj jih je veliko in njihov medsebojni vpliv ni vedno jasen. Za vsakega izmed teh parametrov bi bilo treba najti optimalno vrednost, za katero algoritem najde kar najboljše možne rezultate. Možni so implementacija drugačnega načina izbire, križanja ali mutacije ali pa eksperimentiranje z drugačnimi pogoji za ustavitev genetskega algoritma ter opazovanje vpliva uvedenih sprememb na delovanje algoritma.

Kar zadeva hibridizacije, bi bilo smiselno poskusiti z drugačno lokalno optimizacijo, ki bi optimizirala tudi kandidate, ki ne predstavljajo vozliščnega pokritja, ali pa izvesti obširnejše testiranje vpliva različnih parametrov hibridizacije na potek algoritma.

Vse to so ideje za morebitno nadaljnje delo na tem področju.

Literatura

[1] E. Weisstein. »Vertex Cover Number« From MathWorld–A Wolfram Web Resource. [Online]. Dosegljivo: `http://mathworld.wolfram.com/VertexCoverNumber.html`. [Dostopano 15. 2. 2017].

[2] Wikipedia. »Vertex cover – Wikipedia, the free encyclopedia«, 2016. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Vertex_cover`. [Dostopano 15. 2. 2017].

[3] R. M. Karp. »Reducibility Among Combinatorial Problems«, v zborniku: Complexity of Computer Computations (ur. R.E. Miller, J.W. Thatcher), 1972, str. 85–103.

[4] S. Cook. »The complexity of theorem proving procedures«, *Proceedings of the third annual ACM symposium on Theory of computing*, str. 151–158, 1971.

[5] I. Dinur, S. Safra. »On the hardness of approximating minimum vertex cover« Annals of mathematics, 2005, str. 439–485.

[6] C. H. Papadimitriou, K. Steiglitz. »Combinatorial Optimization: Algorithms and Complexity«, New York: Dover, 1998.

[7] G. Karakostas. »A better approximation ratio for the vertex cover problem«, International Colloquium on Automata, Languages, and Programming, 2005, str. 1043–1050.

[8] M. Karpinski, A. Zelikovsky. »Approximating dense cases of covering problems«, Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilites Location Vol. 40, 1998, str. 169–178.

[9] M. Djordjevic, M. Grgurovič, A. Brodnik. »Performance analysis of the partial use of a local optimization operator on the genetic algorithm for the Travelling Salesman Problem«, v reviji: Business Systems Research Vol. 3 Number 1, 2012, str. 14–22.

[10] S. Tuma. »Kombinatorična optimizacija problema Booleove izpolnjivosti«, Diplomsko delo, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko (2013).

[11] K. Xu. »Benchmarks with Hidden Optimum Solutions for Graph Problems«, 2014 [Online]. Dosegljivo: `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`. [Dostopano: 10. 1. 2017].

[12] A. Dharwadker. »The Vertex Cover Algorithm«, 2006. [Online]. Dosegljivo: `http://www.dharwadker.org/vertex_cover/`. [Dostopano: 10. 1. 2017].

[13] Mathworks. »How the Genetic Algorithm Works - MATLAB & Simulink«, 2017. [Online]. Dosegljivo: `http://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html`. [Dostopano 19. 2. 2017].